한국지능시스템학회 논문지 2009, Vol. 19, No. 3, pp. 437-443

SQL 호스트에서 동적 번호 부여 방식의 성능 평가

Performance Evaluation of Dynamic Numbering Schemes on SQL

홍동권
Dong-Kweon Hong

계명대학교 컴퓨터공학과

요 약

XML 일부분의 변경이 가능한 동적 XML 환경에서 XML의 검색과 변경 기능을 효율적으로 지원하기 위한 방법으로 XML 동적번호 부여 방식에 대한 연구가 활발히 진행되고 있다. 동적 번호 부여 방식은 ORDPATH, DLN과 같이 노드 번호를 부여할 때 인접 노드의 번호에 의존적인 방식과, PSN과 같이 인접한 노드의 번호에 독립적인 노드 번호를 사용하는 2가지의 방식이 있다. 본 논문은 앞의 2가지 동적 번호 부여 방식을 SQL 환경에서 비교하기 위한 환경을 구축하고, W3C XQuery의 변경 형식을 사용하여 그 성능을 비교한다.

키워드 : 동적번호 부여 방식, XML 질의어

Abstract

There has been many active researches on dynamic XML numbering scheme for efficient retrievals and updates of XML. There are two major approaches for dynamic numbering schemes. One approach is dependant on adjacent node numbers when they assign new number for newly inserted node. While the other approach is independent on adjacent node numbers. In this paper we explain the table schema and procedures for our experiments on an SQL host to compare the performance of the two approaches and shows W3C XQuery performance results of the two approaches on the SQL host.

Key Words : Dynamic Numbering Scheme, XQuery, ORDPATH, DLN, PSN

1. 서 론

XML 사용이 점점 더 활성화 됨에 따라 대용량 XML 데이터를 효과적으로 관리하기 위한 많은 방법들이 연구되고 있다. 특히 지금까지 널리 사용되어 온 관계형 데이터베이스 기술을 활용한 XML 데이터의 관리 방안이 그 가능성을 인정받으며 널리 연구되고 있다. 관계형 DB에서 XML을 처리하는 기능은 1)오라클과 같은 DBMS 회사들이 자사의 관계형 DBMS 제품 내부를 변경하는 방법과 2)기존의 관계형 DBMS의 상위 계층에 XML 처리 기능을 추가하는 2가지 방법이 있다. 관계형 모델에서는 계층적인 XML 데이터를 관계형 테이블에 저장할 때 XML의 계층 정보를 저장하는 기법이 필요하며 현재 널리 사용되는 방안은 XML 노드에 적절한 번호를 부여하여 XML 데이터를 관계형 테이블에 저장하는 방식이다. 이때 사용되는 번호 부여 방식은 XML 데이터의 계층 정보를 모두 표현할 수 있어야 하며 XML 질의를 효율적으로 처리할 수 있어야 한다[1, 2, 3, 4].

XML 질의의 표준 활동은 먼저 검색 기능을 중심으로 진행되다가 최근에는 XML 검색 질의에 XML의 변경 기능을 포함하기 위한 형식의 표준화가 진행되고 있다. 따라서 번호 부여 방식도 기존의 정적 환경에서 동적 환경으로의 연구로 바뀌고 있다. 동적 환경에서 XML 트리의 노드 번호가 가져야 할 성질은 다음과 같다[5].

1) XML 트리 내에서 고유한 번호를 가져야 한다.
2) 형제의 순서를 식별할 수 있어야 한다.
3) 노드들 사이의 조상, 자손 관계를 식별할 수 있어야 한다.
4) 노드의 삽입, 삭제, 변경이 효율적으로 이루어져야 한다.

최근에 발표된 인접 번호 의존적인 동적 번호 방식으로는 DLN 기법과 MS사의 SQL 서버에 사용하고 있는 ORDPATH 기법이 있으며[6,7], 인접 번호 비의존적인 방식으로는 PSN 기법이 있다[8]. ORDPATH, DLN은 새로운 노드가 삽입될 때 기존의 노드 번호를 변경시키지 않는 장점을 가지고 있지만 새로운 번호를 생성할 때 인접 번호를 찾아서 참조해야 하는 부담이 있고, PSN은 새로운 번호 생성을 위하여 인접 번호를 찾는 과정은 없지만 형제 노드들의 순서를 유지하기 위하여 형제들 일부분의 정보 변경의 부담이 있다.

본 논문은 기존의 관계형 DBMS를 변경하지 않고 그 상위 레벨에 XQuery 처리 기능을 개발하는 그림 1의 환경에서 인접 번호 의존적인 방식인 ORDPATH와 인접 번호 비의존적인 PSN 방식의 성능을 비교한다.

접수일자 : 2009년 1월 14일
완료일자 : 2009년 4월 4일

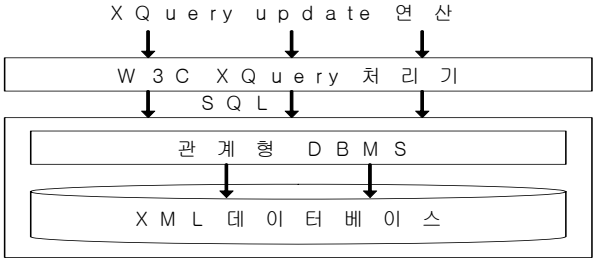


그림 1. XQuery 갱신 처리 환경
Fig 1. XML processing system for XQuery update

2. 관련 연구

2.1 XQuery 동적 연산

XQuery의 동적 연산에는 삽입, 삭제, 변경 기능의 연산이 정의되어 있다. 이 기능들 중에서 가장 중요한 의미를 가지는 삽입 연산의 스펙을 살펴보면 "insert before", "insert after", "insert first", "insert last"로 구성되어 있다. 각각 연산의 의미는 그림 2, 그림 3과 같이 목표 노드를 중심으로 그 삽입 위치가 결정된다.

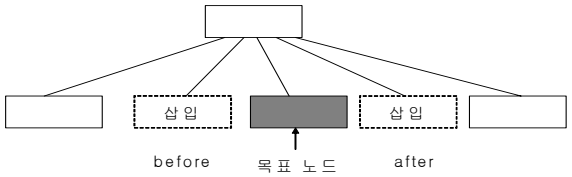


그림 2. “insert before”와 “insert after” 연산
Fig 2. Operations of "insert before" and "insert after"

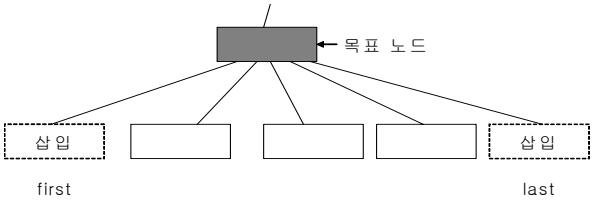


그림 3. “insert first”와 “insert last” 연산
Fig 3. Operations of "insert first" and "insert last"

2.2 ORDPATH 기법

ORDPATH 방식은 MS사에서 개발한 방식으로 그 기본은 Dewey Order[2]의 점 형식을 가지며 레벨에 따라 a.b.c의 형식을 가진다. 이때 a, b, c는 정수 값을 가지며 XML 트리에서 고유한 값을 가진다. XML 트리의 노드에 초기번호를 부여할 때 새로운 노드의 삽입을 고려하여 a, b, c는 홀수 값을 부여 받는다. 그림 4에는 샘플 XML 트리의 ORDPATH 번호를 나타내었다. 초기 번호를 부여할 때에는 홀수만을 사용하므로 모든 노드는 홀수로 적절하게 표현된다. 그림 4에서 부모 자식 관계는 각 레벨의 번호를 이용하여 쉽게 파악할 수 있다.

정리 1: ORDPATH a,b,c에서 a, b, c 가 전부 홀수인 경우 노드 (a.b.c)의 부모는 노드 (a.b)가 되며, 노드 (a.b.c)의 모든 자식들은 n을 임의의 홀수라고 할 때 (a,b,c.n)의 형식을 가진다.

정리 2: 만약 2개의 홀수 x, y에 대해서 $x < y$ 가 성립할 경우 XML 노드의 형제 순서는 (a.b.c.x)가 (a.b.c.y)보다 앞선다.

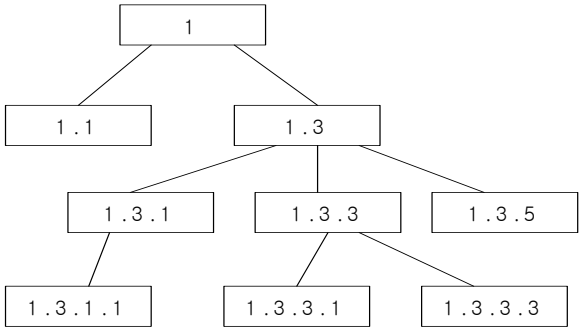


그림 4. 샘플 XML의 초기 ORDPATH 번호
Fig 4. ORDPATH number of sample XML tree

동적 XML 환경에서는 XML 트리의 다양한 위치에 새로운 노드가 삽입될 수 있다. XML 트리 내부에 새로운 노드가 삽입 될 때 임의의 레벨에서 형제 노드의 순서에 따라 다음의 2가지 경우로 나누어진다.
(1) 중간 자식으로 삽입
(2) 양쪽 끝에 삽입

본 논문에서는 양쪽 끝에 삽입하는 단순한 경우는 편의상 생략하고 복잡한 경우인 중간 자식으로 삽입하는 경우만 살펴본다. 그림 4의 XML 트리의 노드 (1.3)의 2번째 자식으로 새로운 노드의 삽입이 발생하는 경우를 살펴보자. 삽입 과정은 그림 5와 같으며 그 결과 XML 트리의 모양은 그림 6과 같다.

```
/* 노드 (1.3)의 2번째 자식으로  노드 추가 */
 노드 (1.3)을 찾는다;
 노드 (1.3)의 1번째와 2번째 자식의
        ORDPATH 번호를 구한다;
 if (삽입한 노드에 부여할 번호가 없으면)
      careting 방식에 의한 새로운 번호를 생성;
 새로운 노드에 ORDPATH 번호 부여;
```

그림 5. ORDPATH 중간 자식으로 삽입 과정
Fig 5. Procedures of insertion ORDPATH as middle child

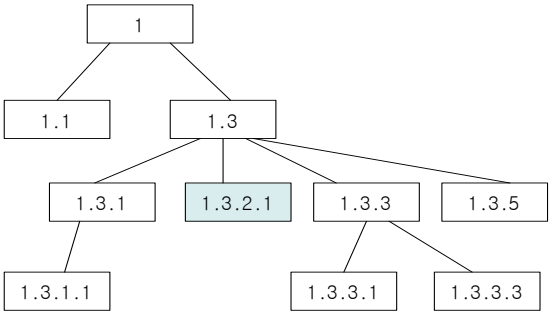


그림 6. ORDPATH 번호의 새로운 노드 추가
Fig 6. Insertion of new ORDPATH number

그림 6에서 새로 추가된 노드는 노드 (1.3.1)과 (1.3.3) 사이에 추가되었다. 새로 추가된 노드는 노드 (1.3.1)과 (1.3.3) 사이의 노드로 $1 < n < 3$ 을 만족하는 (1.3.n)의 형식을 가져야 하는데 적절한 홀수 n 값이 존재하지 않는다. XML 트리의 다른 노드에 전혀 영향을 주지 않기 위하여 ORDPATH는 짝수를 사용하여 새로운 레벨을 추가하는 "careting" 방식을 사용하여 (1.3.2.1)의 번호를 새로운 노드에 부여한다. 따라서 (a.b.c) 형태의 번호에서 짝수는 새로운 노드의 번호를 부여할 때 적절한 번호를 만들어내기 위하여 사용하므로 실제 트리 레벨과는 다른 의미를 가지고 있다.

2.3 PSN 기법

PSN 번호의 형식은 그림 7과 같이 p 필드와 s 필드로 구성된다[8]. p 필드는 부모 자식 관계를 나타내는 필드로 Dewey Order 번호 방식을 채택하며, s 필드는 형제 관계를 나타내며 연속적인 정수를 사용한다.

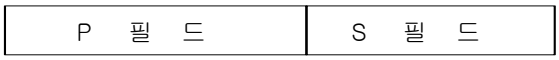


그림 7. PSN의 표현 형식
Fig 7. Representation format of PSN number

각각의 노드는 d 필드에 연속적인 번호를 부여하기 위하여 자식의 s 필드 값의 범위를 나타내는 시작(sno)과 끝(eno) 정보를 가지며, 자식들의 노드를 억세스하지 않고 새로 삽입되는 노드의 p 필드에 고유한 값을 부여하기 위하여 자식의 p 필드 값 중 최대값 (pmax)을 노드의 부가적인 정보로 유지한다. 이런 부가적인 정보들은 질의 최적화에 효과적으로 사용되는 정보로 대부분의 질의 처리 시스템은 최적의 질의 처리 플랜을 만들기 위하여 메타 데이터를 유지한다. PSN 방식으로 표현한 샘플 XML 트리는 그림 8과 같으며 그림을 간단히 표현하기 위하여 sno, eno, pmax 값은 표현하지 않았다. 자식의 p 필드 값은 부모의 s 필드 값에는 영향을 받지 않으며 p 필드 값에만 의존적이다. 초기에는 p 필드 값의 마지막 숫자 값과 s 필드 값이 일치하지만 삽입, 삭제가 발생하면 그 값은 일치하지 않는다. p 필드 값은 항상 XML 트리에서 고유한 값을 가지며 s 필드 값은 음수와 0을 포함한 정수 값을 가진다. PSN 번호가 가지는 성질은 다음의 정리 1, 정리 2와 같다.

정리 1: p 필드 값은 XML 트리에서 고유한 값을 가지며 dewey order의 점 형식을 가지며 레벨에 따라 a.b.c의 형식을 가진다. 이때 a, b, c는 양의 정수 값을 가진다. 그리고 a.b.c의 PSN p 필드 값을 가진 임의의 노드에서 c의 값은 그 부모 노드의 pmax 값보다 크지 않다.

정리 2: 임의의 노드의 s 필드 값은 그 부모 노드의 sno와 eno 사이의 정수 값을 가진다.

따라서 p 필드 값이 고유한 값을 가지므로 앞으로 노드 (a.b.c)라고 하는 것은 p 필드 값을 a.b.c인 노드를 의미한다. 그림 8의 XML 트리의 노드 (1.2)의 2번째 자식으로 새로운 노드의 삽입이 발생하는 경우를 살펴보자.

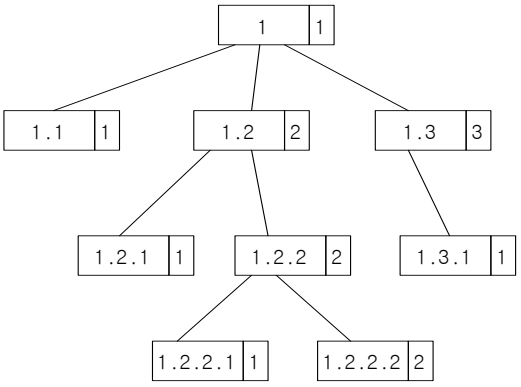


그림 8. 샘플 XML의 PSN 표현
Fig 8. PSN numbers of sample XML tree

```
procedure add()
{
  pare = 부모 노드 (1.2)를 찾는다.
  max = pare.pmax
  새로운 노드의 p 필드 값을 (1.2.max +1)로 부여
                                (1.2.3)
  노드 (1.2.3)은 2번째 자식이 되므로 s 필드 값이
        2가 된다. (1.2.3, 2)
  노드 (1.2.3)의 모든 오른쪽 형제들의 s 필드 값은
        1씩 증가.
  부모 노드의 pmax를 3으로, eno의 값을 1 증가
}
```

그림 9. PSN에서 중간 자식으로 삽입 과정
Fig 9. Procedure of inserting PSN in the middle

그림 9는 삽입 과정이며, 그림 10은 새로운 노드의 삽입 결과이다. 새로 삽입된 노드는 p 필드의 값으로는 1.2.3을 부여 받아 부모 자식의 관계를 잘 나타낸다. 하지만 그 노드가 2번째 자식이 되므로 형제의 순서를 유지하기 위하여 s 필드의 값은 앞 노드 s 필드 값 보다 1 큰 값을 가지게 되며 그 다음에 있는 형제들의 s 필드 값만 1씩 증가된다. 하지만 노드 1.2.2의 자식 노드들은 부모 자식 정보가 바뀌지 않았으므로 PSN 번호가 전혀 바뀌지 않음을 볼 수 있다.

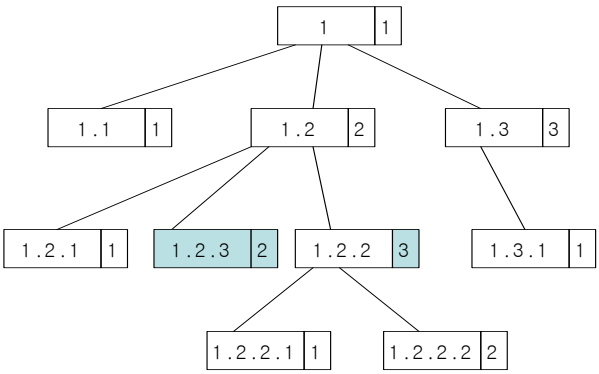


그림 10. 노드 1.2의 2번째 자식으로 노드 삽입
Fig 10. Insertion of a node as 2nd child of node (1.2)

각 노드마다 자식의 시작과 끝 번호를 유지하므로 자식의 개수 n 은 (n = eno - sno + 1) 쉽게 구할 수 있다. n의 값이 큰 경우 새로운 노드의 삽입 위치가 n/2 을 중심으로 왼쪽 또는 오른쪽인지를 검사하고 앞쪽 또는 뒤쪽 형제들의 번호 조정을 할지를 결정한다. 따라서 새로운 노드 삽입 시 평균 n/4 형제들의 s 필드 값 변경만 필요하다. 이때 억세스 된 노드는 부모 노드인 노드 (1.2)와 새로 삽입되는 노드, 그리고 평균 n/4개의 형제 노드들이다.

3. 성능 평가

ORDPATH 방식은 XML 트리의 다른 메타 정보를 사용하지 않는 방식이므로 메타정보를 활용하는 PSN과 비교는 공평하지 못하다. 본 논문에서는 이를 보완하기 위하여 PSN과 같은 조건에서 성능을 비교하기 위하여 ORDPATH에 PSN과 같은 정도의 메타 정보를 사용하는 기능을 추가하여 ORDPATH_EXT라 한다. ORDPATH 방식에 첫 번째 자식, 마지막 자식 그리고 각 노드의 레벨을 저장하는 기능을 추가한 ORDPATH_EXT를 사용하면 XQuery 기능 중에서 "insert first"와 "insert last"를 효과적으로 지원할 수 있게 된다. PSN과 ORDPATH_EXT를 위한 테이블의 구성은 다음과 같다.

ORDPATH_EXT (**ordpath_no**, f_child, l_child, n_level)	
ordpath_no	ORDPATH 방식의 번호 - 스트링
f_child	첫 번째 자식의 ordpath_no - 스트링
l_child	마지막 자식의 ordpath_no - 스트링
n_level	노드의 레벨 (루트 레벨을 1)

그림 11. ORDPATH_EXT 테이블
Fig 11. Table of ORDPATH_EXT

PSN_table (**dewey_no**, sib_ord, pmax, beg_no, end_no, n_level)	
dewey_no	dewey order number 스트링
sib_ord	형제들의 순서를 결정하는 숫자
pmax	자식 중 가장 큰 dewey_no 값
beg_no	자식 중 가장 작은 sib_ord 값
end_no	자식 중 가장 큰 sib_ord 값
n_level	노드의 레벨 (루트 레벨을 1)

그림 12. PSN 테이블
Fig 12. Table of PSN

3.1 ORDPATH_EXT와 PSN의 이론적 비교

ORDPATH_EXT의 경우 (ORDPATH 포함) 새로 입력하는 노드의 번호를 생성하기 위하여 형제 순서로 앞 또는 뒤의 노드를 찾아야 한다. "insert after"의 경우 목표 노드와 목표 노드 다음에 있는 후위 노드의 번호에 따라 새로 삽입되는 노드의 번호가 결정된다. 하지만 목표 노드가 (a.b.n)의 형식을 가질 경우 목표 노드 다음의 후위 노드는 (a.b.*)의 형식의 번호를 가지고 있다는 것 외는 다른 정보가 전혀 없으므로 형제 노드들 중에서 순서를 비교하여 목표 노드 다음의 후위 노드를 찾는 과정이 필요하다.

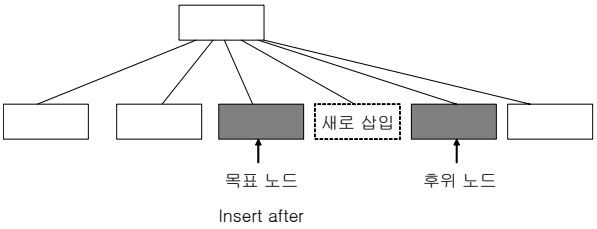


그림 13. Insert after에서 후위 노드의 검색
Fig 13. Finding next node in insert after

목표 노드의 후위 노드를 찾기 위해서는 (a.b.*) 형식의 번호를 가지는 모든 자손들 중에서 목표 노드와 같은 레벨에 있는 형제들을 구분해 낼 수 있는 방법이 필요하다. 모든 자손들을 전부 검색해서 노드 번호를 이용하여 각각의 레벨을 검사하는 것은 비효율적이므로 본 연구에서는 노드의 레벨 (n_level)을 추가하여 사용한다.

ORDPATH_EXT 방식의 경우 새로운 번호를 위하여 같은 레벨에 있는 형제를 읽어 와서 목표 노드의 후위 노드를 찾는 과정이 필요하다. 만약 노드 (1.3.3)과 (1.3.13)이 있을 경우 스트링의 순서로는 (1.3.13)이 더 앞서게 되지만 노드 (1.3.3)이 실제로는 형제 순서로 앞 선 노드이다. 따라서 형제들의 순서는 각 필드를 숫자로 바꾸어 비교하는 과정이 필요하다.

3.2 모의 실험 성능 평가

본 논문의 모의 실험 평가에서는 XQuery의 변경 기능 중에서 가장 복잡한 기능을 보이는 "insert after"의 성능을 평가한다.

3.2.1 실험 데이터 생성

샘플 데이터로 사용된 샘플 XML 트리는 트리의 깊이 (depth)와 차수(degree)를 변경하면서 실험하였다. 본 논문에서는 실험 결과를 가장 잘 보여주는 값을 선정하여 본 실험에서는 XML 트리의 레벨은 5, 단말 노드를 제외한 모든 노드들은 각각 20개씩의 자식 노드들을 가지고 있고 노드들의 총 개수는 168,421 XML 트리를 사용하였다.

3.2.2 ORDPATH_EXT 프로시져의 구성

"insert after(target_node)" 형식의 입력을 ORDPATH_EXT 방식으로 처리하기 위한 프로시져는 다음과 같다. 더 효율적인 방법이 있을 수 있으나 본 논문에서 사용한 방법은 먼저 목표 노드보다 큰 값을 가진 것들로 뷰를 만들고, 뷰에서 가장 작은 값을 가진 노드를 찾는 방식을 사용한다.

그림 14의 SQL 프로시져에서 사용한 함수 o_level_func은 ORDPATH 방식의 노드 번호에서 노드의 레벨을 구하는 함수이며, compare_value의 기능은 2개의 ordpath_no를 비교하여 2번째 인자가 클 경우 참(TRUE)을 반환하는 함수이다. compare_value 함수는 다음의 노드 번호가 있을 경우 각 필드별 값을 비교하여 다음과 같은 순서를 유지한다.

(1.3.1) < (1.3.2.1) < (1.3.2.2.1) < (1.3.2.3) < (1.2.13) < (1.3.3) < (1.3.13)

```
CREATE VIEW GTNODE AS
SELECT ordpath_no
FROM ORDPATH_EXT
WHERE n_level = o_level_func(목표노드)
   AND compare_value(목표노드, ordpath_no);

SELECT ordpath_no
FROM GTNODE
MINUS
SELECT S.ordpath_no
FROM GTNODE S, GTNODE T
WHERE compare_value(T.ordpath_no,
S.ordpath_no);
```

그림 14. ORDPATH_EXT의 Insert_after 프로시져
Fig 14. Insert_after procedure of ORDPATH_EXT

3.2.3 PSN 프로시져의 구성

PSN 방식으로 "insert after(target_node)"를 구현할 경우 목표 노드 다음의 후위 노드의 노드 번호를 참조할 필요는 없다. 다만 새로 삽입되는 노드의 뒤에 있는 형제들의 S 필드의 값을 변경하기 위한 과정이 필요하다. 목표 노드의 sib_ord 값을 N 이라고 할 경우 다음의 SQL 구문으로 삽입된 노드 다음에 존재하는 형제 노드들의 sib_ord 값을 변경할 수 있다.

```
SELECT sib_ord INTO N
FROM psn_table
WHERE dewey_number = target_node;

UPDATE PSN_TABLE
SET sib_ord = sib_ord + 1
WHERE n_level = (p_level_func(target_node) +1)   AND
sib_ord > (N +1);
```

그림 15. PSN 방식의 insert_after 프로시져
Fig 15. Insert_after procedure of PSN

그림 15의 프로시져에서 p_level_func는 목표 노드의 노드 레벨을 계산하는 함수이다. PSN 방식에서 형제들의 s 필드 값을 변경하는 것의 비교, S 필드 값은 숫자로 되어 순서의 비교가 쉽다.

3.2.4 실험 결과

ORDPATH와 PSN의 번호를 10진수로 표현한 실험 결과에서 ORDPATH가 비정상적으로 많은 실험 시간을 소비하여 정상적인 평가가 어려웠다,

4.비트 스트링 표현 평가

ORDPATH와 PSN 2가지 번호 부여 방식을 10진수로 사용했을 때 ORDPATH 방식의 정렬 시간이 너무 오래 걸려 ORDPATH 경우 번호 표현 방식을 ORDPATH 방식에서 제시한 "prefix free encoding" 방식을 적용하여 다음의 그림 16과 같이 표현하여 다시 실험하였다.

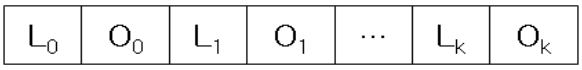


그림 16. ORDPATH 방식의 비트스트링 표현
Fig 16. Bit string representation of ORDPATH number

본 실험에서는 ORDPATH, PSN 방식으로 번호를 생성하여 데이터베이스 테이블에 저장한 뒤, 임의의 노드 바로 다음에 새로운 노드를 추가하는데 소요되는 시간을 측정하였다. 여기서 새로운 노드를 추가하는데 걸리는 시간은 삽입할 위치를 찾는 시간, 삽입 시간, 그리고 새로운 노드의 삽입으로 인해 기존 노드들의 순서 정보들을 업데이트하는 시간들을 모두 포함하였다.

본 실험에서 구현한 모든 프로시져는 오라클의 PL/SQL로 작성하였다.

〈ordpath 방식〉

우선 ordpath 방식을 기반으로 새로운 노드의 삽입을 처리하기 위해 필요한 프로시져들을 다음과 같이 6가지이며 본 실험에서는 이를 모두 구현하여 실험을 평가하였다.

ordpath_insert_after(target_node)

이 프로시져는 입력으로 주어진 target_node 다음에 새로운 노드를 생성하여 추가한다. 예를 들어 ordpath_insert_after('1.3') 경우 '1.3' 노드 다음에 새로운 노드를 생성하여 추가한다. (이 경우, 처음으로 삽입한다면 해당 프로시져에 의해 '1,3' 다음에는 '1.4.1'노드가 생성 및 추가된다.)

prefix_encoding(dewey_number)

이 프로시져는 ordpath의 dewey 번호를 마이크로소프트사에서 제안한 prefix 방식으로 인코딩하여 일련의 이진수로 변환하여 리턴한다.

prefix_decoding(binary_number)

prefix_encoding 프로시져와 반대로 prefix 방식으로 인코딩된 이진수를 다시 dewey 번호로 변환하여 반환한다.

dot_length(node)

dewey 번호에서 '.'의 개수를 반환한다.

bitstring(decimal_number)

십진수를 이진수로 변환한다.

decimalstring(bitstring)

이진수를 십진수로 변환한다.

〈psn 방식〉

psn 방식으로 노드의 삽입을 처리하는 방법은 간단하며 다음과 같이 1개의 프로시져로 처리할 수 있다.

psn_insert_after(target_node)

ordpath_insert_after와 동일하게 입력으로 주어진 노드의 바로 다음 위치에 새로운 노드를 삽입한다.

4.1 측정 방법

본 실험에서는 소수개의 노드 삽입 시 매우 빠른 실행시

간으로 인한 시간 측정에 따른 어려움으로 인해 특정 위치에 새로운 노드들은 연속적으로 약 100번 삽입하여 걸리는 시간을 측정한 뒤 이를 토대로 1개의 노드를 삽입할 경우 걸리는 평균 시간을 구하였다. 그리고 2가지 방식 모두의 성능 향상을 위해 psn_table 테이블의 dewey_no 컬럼에 인덱스를 생성한 반면 ordpath_ext 테이블에는 어떠한 인덱스도 두지 않았다. ordpath_ext 테이블의 prefix_num 컬럼에 인덱스를 둘 경우 삽입시 형제 노드를 찾기 위해 사용되는 SQL문의 where 조건에 의한 prefix_num 컬럼의 결과값의 분포도가 현저히 높으므로 인덱스 생성으로 인해 성능이 오히려 저하된다. 실제로 실험을 통해 ordpath_ext 테이블에 인덱스를 두었을 경우 그렇지 않을 때 보다 2배 이상으로 실행 시간이 오래 걸렸다. 따라서 본 실험에서는 2가지 방식 모두 보다 나은 성능을 발휘 할 수 있도록 psn_table 테이블에는 인덱스를, 반면 ordpath_ext 테이블에는 어떠한 인덱스도 두지 않았다.

위의 실험 환경을 바탕으로 PSN과 ORDPATH 방식을 이용하여 '1.3.1' 노드 다음에 새로운 노드의 삽입을 아래의 프로시져로 연속 100회 실시하였으며 또한 이러한 작업을 40회 실시하여 회마다 100개씩 증가되는 형제 노드(1.3.1의 형제노드)들의 총 개수에 따라 새로운 노드의 평균 삽입 시간이 어떻게 변하는가를 측정하였다.

```
declare
  i number;
begin
  for i IN 1..100 Loop
      프로시져('1.3.1');
  END loop;
end;
```

4.2 실험 결과

형제 노드들의 개수에 따른 새로운 노드의 평균 삽입 시간을 그래프로 나타내면 다음과 같다.

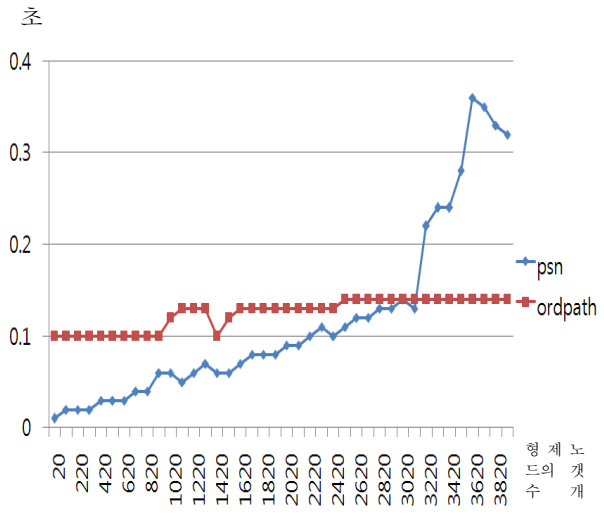


그림 17. 형제 노드 개수의 변화에 따른 성능의 비교
Fig 17. Performance comparison with changes of the number of siblings

그림 17의 그래프에서 X축의 형제 노드의 개수는 본 실험에서의 목표 노드의 모든 형제 노드들, 즉 1.3.1 노드의 형제 노드들의 총 개수를 말한다. 초기에는 단말 노드를 제외한 모든 각각의 노드들이 20개씩의 자식 노드를 가졌다. 1.3.1 노드 다음에 새로운 노드를 연속으로 100회 실행함에 따라 1.3.1의 형제노드의 개수는 120으로 되었으며 이때 1개의 노드를 삽입할 때 소요되는 평균 시간은 PSN 방식의 경우 약0.02초, ORDPATH 방식의 경우 약 0.1이다. 목표노드의 형제 노드들의 개수가 20에서 3000개 이하일 경우 PSN이 ORDPATH 방식보다 훨씬 우수한 성능을 보이고 있다. 반면에 형제 노드들의 개수가 3000개 이상으로 늘어남에 따라 PSN 방식이 약0.2초에서 계속적으로 증가한다. 하지만 ORDPATH 방식은 여전히 약0.14초를 유지함에 따라 노드들의 개수가 많을 경우 ORDPATH가 더 나은 성능을 보이고 있다.

4.3 실험 결과 분석

ORDPATH 방식의 경우 아래의 SQL문으로 삽입할 지점 바로 뒤에 오는 형제 노드를 ordpath_ext 테이블로부터 검색하는 것이 대부분의 시간을 소비하는 주요 작업이라고 볼 수 있다.

select min(prefix_num) from ordpath_ext where prefix_num > prefix_인코딩된_목표노드 and 목표노드의_level = level_num;

즉 ordpath_inser_after('1.3.1') 경우 새로운 노드를 삽입하기 위해 현재 1.3.1' 다음에 위치한 형제노드를 찾아야 되는데 이를 위해서는 1.3.1노드의 모든 형제노드들을 prefix 인코딩된 값들을 기준으로 정렬한 뒤 그 중에서 가장 작을 값을 추출한다. PSN 방식의 경우 ORDPATH 방식과는 달리 부모노드의 dewey 번호와 형제들 간의 순서 번호가 분리되어 있으므로 이를 이용하여 형제 노드들을 쉽게 검색할 수 있다. 하지만 아래의 SQL문으로 새로운 노드의 삽입으로 인해 나머지 형제노드들의 S 필드의 순서 정보가 모두 바뀌어야 한다.

update psn_table set sib_ord = sib_ord+1 where dewey_no like 목표노드의_부모노드 and n_level = 목표노드의_level and sib_ord >= 목표노드의_sid_ord;

ORDPATH 방식이 이웃하는 형제노드를 검색하기위해 정렬하는데 많은 노력이 필요하다면 PSN 방식의 경우 삽입된 지점 이후의 모든 형제 노드들의 S 필드의 값을 업데이트 하는데 많은 시간이 소요된다. 위의 실험 결과를 종합해 보면 삽입할 지점의 형제 노드들의 개수가 적을수록 PSN 방식이 훨씬 유리함을 알 수 있다. 즉, PSN 방식의 경우 정렬작업이 필요 없으며 형제 노드들의 개수가 적은 경우 그들의 개수만큼 s 필드 값을 업데이트 해주면 되기 때문이다. 하지만 형제 노드들의 개수가 많은 경우, 이는 업데이트해야할 노드들이 많다는 것을 의미하므로 새로운 노드의 삽입으로 인한 실행 시간이 지연되게 된다. ORDPATH의 경우 처음에는 PSN 보다 나쁜 성능을 보이지만 PSN에 비해서 형제 노드들의 개수에 크게 영향을 받지 않으며 비교적 초기 실행 시간을 유지하는 것을 확인할 수 있다.

5. 결 론

XML 일부분의 변경이 가능한 동적 XML 환경에서 연구된 동적 번호 부여 방식은 ORDPATH와 같이 노드 번호를 부여할 때 인접 노드의 번호에 의존적인 방식과 PSN과 같이 인접한 노드의 번호에 독립적인 노드 번호를 사용하는 2가지의 방식이 제안되었다. 본 논문은 앞의 2가지 동적 번호 부여 방식을 SQL 환경에서 비교하여 그 장단점을 알아보았다.

먼저 MS사의 특허 방식인 ORDPATH는 번호를 특별한 prefix free 비트스트링으로 표현하여야 하며, XML 트리의 형제 노드의 개수에 민감하지 않은 성능을 나타내었다. 하지만 번호의 표현과 구현이 어려운 단점이 있다. 하지만 PSN 방식은 ORDAPATH 방식에 비해서 형제 노드의 개수가 3,000개 이하인 경우 매우 우수한 성능을 보였다. 따라서 형제 노드의 개수가 3,000 이하인 많은 응용에서는 PSN과 같이 인접번호 비의존적인 방식이 훨씬 효과적임을 알 수 있다.

참 고 문 헌

[1] J. Shanmugasumdaram et al, "Relational Databases for Querying XML document: Limitations and Opportunities" *in Proceedings of the 25th VLDB Conference,* pp 302-314 1999.

[2] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, G. Lohman, "On supporting Containment Queries in Relational Database Management Systems" *in Proceedings of ACM SIGMOD, May Santa Barbara, pp 425-436 CA* 2001.

[3] D. Dehan, D. Toman. M. Consens, and M. Tamer Ozsu, "A Comprehensive XQuery to SQL Translation using Dynamic Interval Encoding" *in Proceedings of ACM SIGMOD, San Diego pp 623-634 CA,* 2003.

[4] I. Tatarinov, S. Viglas, K.Bayer, J. Shanmugasundaram, E. Shekita, C. Zhang, "Storing and Querying Ordered XML Using a Relational Database System" *in Proceedings of ACM SIGMOD pp 204-214* 2002.

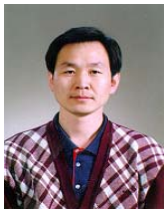
[5] T. Bohme, E. Rahm, "Supporting Efficient Streaming and Insertion of XML data in RDBMS" *in Proceeding of CaiSE'04 Workshop,* Volume 3(DIWeb'04), pp70-81, 2004.

[6] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, "ORDPATHs: Insert-friendly XML node lables" *in Proceeding of ACM SIGMOD,* pp 903-908 June Paris, France 2004.

[7] T. Harder, M. Haustein, C. Mathis, M. Wagner, "Node labeling schemes for dynamic XML documents reconsidered" *Data and Knowledge Engineering* Volume 60, Issue 1, 2007.

[8] D. Hong, "PSN: A dynamic numbering scheme for XQuery Update Facility" *International Journal of Fuzzy Logic and Intelligence Systems*, June 2008.

저 자 소 개



홍동권(Dong-Kweon Hong)

1985년 : 경북대학교 전자과 공학사
1992년 : U. of Florida 컴퓨터공학 석사
1995년 : U. of Florida 컴퓨터공학 박사
1997년～현재 : 계명대학교 컴퓨터공학과 교수

관심분야 : XML, 데이터베이스, 질의 최적화
Phone : 053-580-5281
Fax : 053-580-5165
E-mail : dkong@kmu.ac.kr